Autores

Ignacio Martín Franco UO239718 71727308Z  
Grabriela Agnieszka Lagowska UO231379 X9120238W

DOcuMENTACIÓN  
Red social

Sdi1-UO231379UO239718

Índice

[Una descripción clara y detallada de la implementación de cada uno de los casos de uso implementados. 2](#_Toc508983314)

[1 Público: registrarse como usuario 2](#_Toc508983315)

[2 Público: iniciar sesión 2](#_Toc508983316)

[3 Usuario registrado: listar todos los usuarios de la aplicación 2](#_Toc508983317)

[4 Usuario registrado: buscar entre todos los usuarios de la aplicación 2](#_Toc508983318)

[5 Usuario registrado: enviar una invitación de amistad a un usuario 3](#_Toc508983319)

[6 Usuario registrado: listar las invitaciones de amistad recibidas 3](#_Toc508983320)

[7 Usuario registrado: aceptar una invitación recibida 4](#_Toc508983321)

[8 Usuario registrado: listar los usuarios amigos 4](#_Toc508983322)

[9 Usuario registrado: crear una nueva publicación 6](#_Toc508983323)

[10 Usuario registrado: listar mis publicaciones 6](#_Toc508983324)

[Un catálogo de las pruebas unitarias realizadas y descripción sencilla de cada una 7](#_Toc508983325)

[1.1 [RegVal] 7](#_Toc508983326)

[1.2 [RegInval] 7](#_Toc508983327)

[2.1 [InVal] 7](#_Toc508983328)

[2.2 [InInVal] 7](#_Toc508983329)

[3.1 [LisUsrVal] 7](#_Toc508983330)

[Acceso al listado de usuarios desde un usuario en sesión 7](#_Toc508983331)

[3.2 [LisUsrInVal] 7](#_Toc508983332)

[4.1 [BusUsrVal] 7](#_Toc508983333)

[4.2 [BusUsrInVal] 7](#_Toc508983334)

[5.1 [InvVal] 7](#_Toc508983335)

[5.2 [InvInVal] 7](#_Toc508983336)

[6.1 [LisInvVal] 7](#_Toc508983337)

[7.1 [AcepInvVal] 8](#_Toc508983338)

[8.1 [ListAmiVal] 8](#_Toc508983339)

[9.1 [PubVal] 8](#_Toc508983340)

[10.1 [LisPubVal] 8](#_Toc508983341)

[11.1 [LisPubAmiVal] 8](#_Toc508983342)

[11.2 [LisPubAmiInVal] 8](#_Toc508983343)

[12.1 [PubFot1Val] 8](#_Toc508983344)

[12.1 [PubFot2Val] 8](#_Toc508983345)

[13.1 [AdInVal] 8](#_Toc508983346)

[13.2 [AdInInVal] 8](#_Toc508983347)

[14.1 [AdLisUsrVal] 8](#_Toc508983348)

[15.1 [AdBorUsrVal] 8](#_Toc508983349)

[15.2 [AdBorUsrInVal] 8](#_Toc508983350)

[Cualquier otra información necesaria para una descripción razonablemente detallada de lo entregado y su correcto despliegue y ejecución. 9](#_Toc508983351)

# Una descripción clara y detallada de la implementación de cada uno de los casos de uso implementados.

Para implementar los casos de uso se ha seguido el patrón arquitectónico visto en clase. Los elementos necesarios para realizar la entrega son los siguientes:

* Entidades:
* User
* Publication
* Request
* Controladores:
* UsersContoller
* PublicationsController
* RequestsController
* FriendsController
* HomeController
* Servicios:
* InsertSimpleDataService
* PublicationsService
* RequestsService
* SecuriryService
* UsersDetailsServiceImpl
* UsersService
* Repositorios:
* PublicationsRepository
* RequestsRepository
* UsersRepository
* Vistas

## A continuación veremos en detalle el uso de cada uno de los elementos en los casos de uso.

## 1 Público: registrarse como usuario

Para implemetar este caso de uso se han utilizado los componentes **User, UsersRepository, UsersController, UsersService** , la vista **signup**.**html** y una clase adicional denominada **SignUpFormValidator** para validar los datos introducidos por los usuarios a la hora de registrarse.

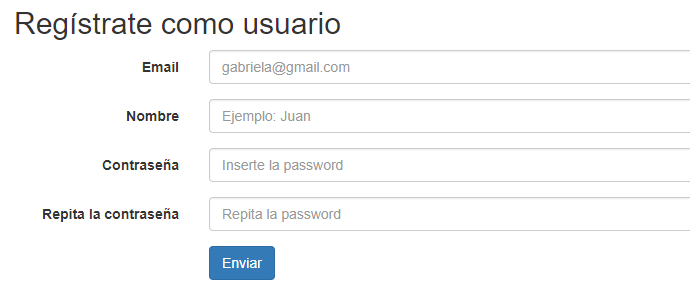
Tras definir los atributos necesarios en la clase **User** se ha implementado el método **addUser** en UsersService .Este método hace uso del método save(…) del repositorio UsersRepository para guardar el usuario y permite encriptar la contraseña de éste gracias a inyectar el bean BCryptPasswordEncoder en el servicio.

A continuación se han implementado los métodos GET y POST en el controlador para responder a las peticiones de los usuarios. El método GET devuelve la vista signup y el método POST utiliza el servicio usersService para guardar el nuevo usuario.

Posteriormente se ha creado el validador denominado **SignUpFormValidator** para hacer todas las comprobaciones necesarias sobre los datos que ha proporcionado un usuario en el formulario de registro.

Por último se ha inyectado el bean validador en el controlador. Una vez hecho esto, en el método GET que muestra la vista del formulario se ha incluido una entidad vacía en el modelo enviado a la vista, se ha añadido la entidad vacía en la plantilla, se ha modificado el método POST añadiendo la anotación **@Validated** en la cabecera del método para indicar el atributo del modelo a validar (en nuestro caso de tipo User) y también se ha añadido el objeto **BindingResults** que contiene los resultados de la validación y define el método **hasErrors** que indica si se produjo algún error en el registro.

El resultado de implementar este caso de uso es el siguiente:



## 2 Público: iniciar sesión

Para que un usuario registrado pueda iniciar sesión en la aplicación se han utilizado los siguientes elementos: **UsersReposiory, UsersService, UserDetailsServiceImpl, SecurityService, UsersController, WebSecurityConfig** y la vista **login.html**.

El email es lo que nos permite identificar al usuario en el login, por tanto se ha creado en primer lugar el método **User findByEmail(String email)** en el repositorio. Una vez definido este método, añadimos un método en **UsersService** que llama al método **findByEmail** del repositorio **UsersRepository**.

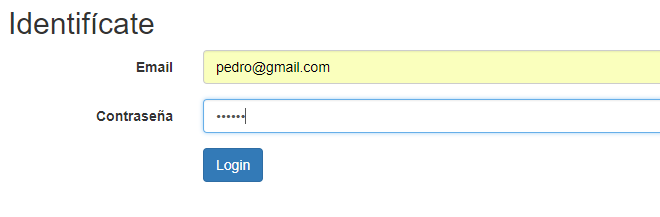
A continuación, se ha modificado la clase **UserDetailServiceImpl** para obtener el usuario de la aplicación y crear un **userDetails** con email y password.

Posteriormente se ha creado la clase **SecurityService** con dos métodos, uno para devolver el usuario actual autenticado y otro para permitir el inicio automático de sesión después de que un usuario cree una cuenta.

Una vez hecho esto se ha modificado la clase **UsersControllers**. Se ha inyectado el servicio **securityService** y, además se han creado los métodos GET **login() y home()** para mostrar sus vistas correspondientes.

A continuación, se ha creado la configuración **WebSecurityConfig** en la que se definen las autorizaciones. En caso del login establecemos la página después de una autenticación válida y otra página en caso de auntenticación inválida. Dado que en caso de una **autenticación inválida** tenemos que mostrar qué error hubo, definimos la siguiente redirección: failureUrl("/login?error=true"). Dado que esta URL contiene un parámetro, se ha definido un parámetro adicional en el método GET del login para que en caso del usuario introduzca datos incorrectos al iniciar sesión se le muestre un error.

Por último creamos la vista login en la que incluimos un campo para poder mostrar el error cuando lo haya.



## 3 Usuario registrado: listar todos los usuarios de la aplicación

Para obtener la lista de todos los usuarios de la aplicación se han utilizado los siguientes elementos: **UsersRepository, UsersService, UsersController** y la vista **list.html.**

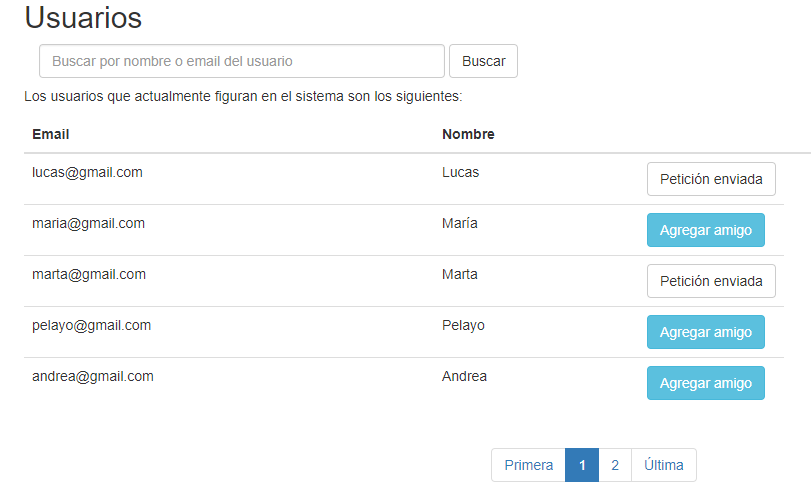
En primer lugar se ha creado el método Page<User> **findAllButUser**(Pageable pageable, User user) que devuelve una lista paginada de todos **los usuarios de la aplicación menos el usuario autenticado**.

A continuación, se ha definido el método **getUsersForUser** en **UsersService** que llama al método mencionado anteriormente del repositorio.

Posteriormente se ha definido el método **getListado** en el controlador. Este método obtiene el usuario autenticado y utiliza el servicio userService para obtener la **lista paginada de usuarios** y retorna la vista user/list.

Por último se ha creado la vista **list.html** en la que se incluye el fragmento **pagination.html** que permite incluir la paginación.

Así pues, tras iniciar sesión con algún usuario podemos acceder a la opción del menú llamada “**Ver** **usuarios**” donde se nos muestra la lista con todos los usuarios de la aplicación menos el usuario en sesión.



## 4 Usuario registrado: buscar entre todos los usuarios de la aplicación

Para que un usuario pueda buscar entre todos los usuarios de la aplicación se han utilizado los siguientes elementos: **UsersRepository, UsersService, UsersController** y la vista **list**.**html**.

Para empezar, se ha definido un método nuevo en **UsersRepository** llamado **searchByNameOrEmail** que contiene un parámetro String **search** y retorna una lista paginada de usuarios para el usuario en sesión cuando el texto buscado coincide con el nombre o email de algún usuario de la aplicación.

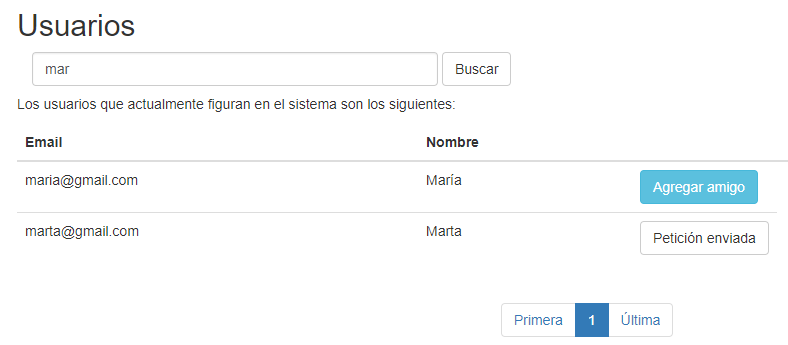
A continuación se ha imlementado el método **searchUsersByNameForUser** en **UsersService** que llama al método **searchByNameOrEmail** del repositorio. En este método se ha hecho uso de comodines SQL para comprobar si la cadena buscada se encuentra en el nombre o email de algún usuario.

Tras hacer esto, se ha modificado el método **getListado** de **UsersController** añadiéndole el parámetro String **search** para que en caso de que éste sea distinto de null se llame al método **searchUsersByNameForUser** del servicio o en caso contrario devolver la lista de todos los usuarios de la aplicación menos el usuario en sesión.

Por último se ha incluido un **formulario de búsqueda** en la vista list.html

Por tanto, tras **iniciar sesión** con algún usuario y acceder a la opción “**Ver usuarios**”, donde se nos muestra la lista con todos los usuarios de la aplicación **menos el usuario en sesión**, podemos **buscar al usuario** que nos interesa introduciendo una cadena en el campo de búsqueda.

Si introducimos la cadena “mar” se nos van a mostrar todos los usuarios cuyo email o nombre contenga la cadena introducida.



## 5 Usuario registrado: enviar una invitación de amistad a un usuario

Una vez seguidos los pasos del caso de uso anterior (4 Usuario registrado: buscar entre todos los usuarios de la aplicación) se podrá enviar una invitación de amistad a aquellos usuarios que tengan asociados el **botón con el mensaje “Agregar amigo”.** Al hacer click el valor del botón **pasará a ser “Petición enviada”.**



Para que un usuario pueda enviar solicitudes de amistad se han usado: **RequestRepository, UsersService, RequestService, UserController y RequestController** junto con las la vista “**user/list.html”.**

Para empezar, se ha definido un repositorio “**RequestRepository**” para manejar las invitaciones de amistadad. Al pulsar el botón “**Agregar amigo**” se llama al método **createRequest** del controlador “**RequestController**”. Este método **crea un objeto “Request**” y se le asigna como usuario “maker” al usuario en sesión, y usuario “target” el usuario asociado al botón. Después **actualiza** lista de usuarios.

## 6 Usuario registrado: listar las invitaciones de amistad recibidas

Una vez registrado el usuario válido (en este caso **pedro@gmail.com** con contraseña **123456**) se hace click sobre “**Peticiones** **recibidas**”:



A continuación, se mostrará una lista de las peticiones recibidas que tiene el usuario.



Los elementos usados son: **UserService, RequestService y RequestController** junto con las la vista “**request/list.html**”.

Se ha definido un método en **RequestService** llamado **getRequestsForUser** que devuelve todas las **peticiones que tiene pendientes un usuario** (normalmente para usuario en Sesión)

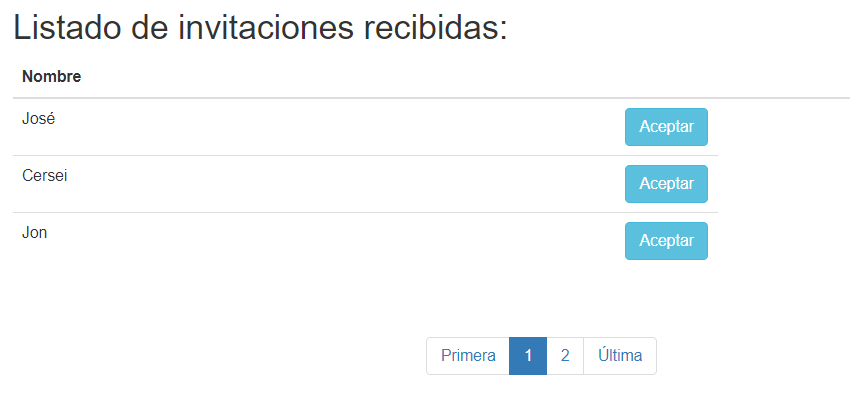
## 7 Usuario registrado: aceptar una invitación recibida

Siguiendo el caso de uso anterior (6 Usuario registrado: listar las invitaciones de amistad recibidas).

Al hacer click en el botón “**Aceptar**” de algún usuario, el controlador **RequestController** recoge mediante la anotación @RequestMapping("request/accept/{id}") **public** String getListado el **id de la petición**. A continuación:

1. Se añaden mutuamente como amigos a sus listas de amigos
2. Se elimina petición de la lista del usuario en sesión
3. Se elimina la petición y también si existía la misma petición, pero en dirección inversa (usuario en sesión mandó petición de amistad)

Tras aceptar a “Andrea”, “María” y “Sansa” **las invitaciones desaparecerán**, serán amigos y la lista quedará:



## 8 Usuario registrado: listar los usuarios amigos

Una vez registrado el usuario válido (en este caso **pedro@gmail.com** con contraseña **123456**) se hace click sobre “**Ver usuarios**” y comprobamos que no tiene amigos:



Tras ir a “**Invitaciones recibidas**” y **aceptar** todas las invitaciones recibidas (de “Andrea”, “María”, “José”, “Sansa”, “Cersei” y “Jon”)



Y en la vista “**Mis amigos**”:



Si cerramos sesión haciendo click en “**Desconectar**” e iniciamos sesión con **andrea@gmail.com** comprobaremos que tiene como amigo a “Pedro”:





Y en el link “**Mis amigos**” mostrará como amigo a “**Pedro**”:



Los elementos usados son “**RequestController**”, “**UserController**”, “**RequestService**”, “**UserService**” y los repositorios correspondientes además de las vistas “**friend**/**list**.**html**”, “**user**/**list**.**html**” y “**request/list**.**html**”.

**En** **un** **inicio**, **ningún** **usuario** **tiene** **amigos**, por lo que tendrá la lista de amigos vacía. Para hacer amigos la única manera es **aceptando** **invitaciones** de amistad o que acepten alguna tuya (definido en caso de uso 7: aceptar una invitación recibida).

## 9 Usuario registrado: crear una nueva publicación

Para implemetar este caso de uso se han utilizado los componentes **Publication, PublicationsRepository, PublicationsController, PublicationssService** , la vista **addPublication**.**html** y una clase adicional denominada **RegisterPublicationFormValidator** para validar los datos introducidos por los usuarios a la hora de crear la publicacion.

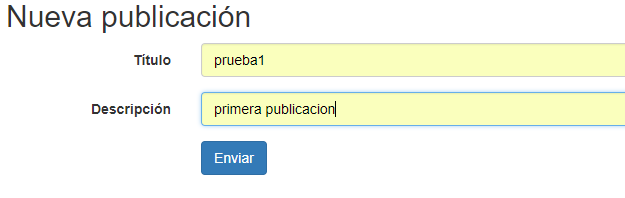
Tras definir los atributos necesarios en la clase **Publication** se ha establecido una **relación** entre esta entidad y la entidad **User** ya que un usuario puede tener una o más publicaciones.

Después se ha implementado el método **addPublication** en PublicationsService..Este método hace uso del método save(…) del repositorio PublicationsRepository para guardar la publicación.

A continuación se han implementado los métodos GET y POST en el controlador para responder a las peticiones de los usuarios. El método GET devuelve la vista publication/addPublication y el método POST obtiene el usuario en sesión, utiliza el servicio publicationsService para **guardar la publicación** de dicho usuario y devuelve la vista correspondiente.

Posteriormente se ha creado el validador denominado **RegisterPublicationFormValidator** para hacer todas las comprobaciones necesarias sobre los datos que ha proporcionado un usuario en el formulario de registro de la publicación.

Por último se ha inyectado el bean validador en el controlador. Una vez hecho esto, en el método GET que muestra la vista del formulario se ha incluido una entidad vacía en el modelo enviado a la vista, se ha añadido la entidad vacía en la plantilla, se ha modificado el método POST añadiendo la anotación **@Validated** en la cabecera del método para indicar el atributo del modelo a validar (en nuestro caso de tipo Publication) y también se ha añadido el objeto **BindingResults** que contiene los resultados de la validación y define el método hasErrors que indica si se produjo algún error en el registro.



El usuario que ha iniciado sesión puede **crear una nueva publicación** accediendo a la opción del menú “Publicaciones” – “Crear publicación”. Tras crear la publicación que podemos ver en la captura anterior, se nos redirige a la vista que contiene la lista con todas las publicaciones del usuario en sesión.

## 10 Usuario registrado: listar mis publicaciones

Para obtener la lista de las publicaciones de un usuario de la aplicación se han utilizado los siguientes elementos: **PublicationsRepository, PublicationsService, PublicationsController** y la vista **listPublicatios.html.**

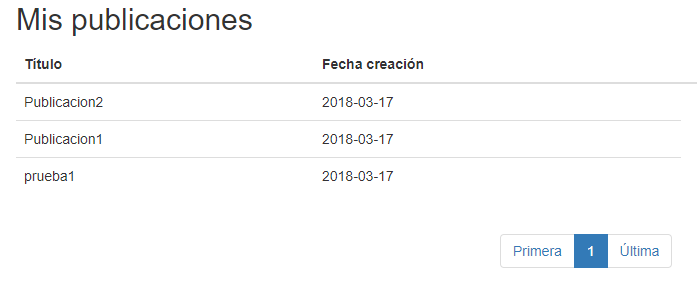
En primer lugar se ha creado el método Page<Publication> **findAllByUser**(User user,Pageable pageable) que devuelve una **lista paginada de todas las publicaciones del usuario** en sesión.

A continuación se ha definido el método **getPublicationsForUseren** PublicationsService que llama al método mencionado anteriormente del repositorio.

Posteriormente se ha definido el método **getList** en el controlador. Este método obtiene el usuario autenticado y utiliza el servicio **publicationsService** para obtener la lista paginada de publicaciones y retorna la vista publication/listPublications.

Por último se ha creado la vista **listPublications.html** en la que se incluye el fragmento pagination.html que permite hacer uso de la paginación.

El usuario en sesión puede ver todas sus publicaciones accediendo a la opción del menú “**Publicaciones**”- “**Mis** **publicaciones**”



## 11 Usuario registrado: listar las publicaciones de un usuario amigo.

## 

Para obtener la lista de las publicaciones de un usuario amigo se han utilizado los siguientes elementos: **PublicationsRepository**, **PublicationsService**, **PublicationsController** y la vista **listPublicatios.html**.

Dado que este caso de uso es muy parecido al anterior, lo único que se ha tenido que hacer fue modificar la vista **lista**.**html** de los usuarios amigos y convertir **el campo nombre en un enlace** de la siguiente manera:

th:href=*"${'/publication/publicationsFriend/' + user}*

de este modo cuando pinchemos en el nombre de algún usuario amigo se nos **redirige** a la url anterior pasando como parámetro el usuario.

A continuación se ha añadido **un nuevo método en el controlador** que contiene el parámetro user en la URL y hace uso del servicio publicationsService para obtener la lista paginada del usuario pasado por parámetro.

Para que un usuario en sesión pueda ver las publicaciones de un usuario amigo tiene que acceder a la opción del menú “**Mis** **amigos**” y pinchar en el nombre del amigo que quiera.

# Un catálogo de las pruebas unitarias realizadas y descripción sencilla de cada una

## [RegVal]

Registro de Usuario con datos válidos.

**1-Vamos al formulario de registro**

PO\_HomeView.*clickOption*(*driver*, "signup", "id", "btnRegistro");

**2-Rellenamos el formulario.**

PO\_RegisterView.*fillForm*(*driver*, "carla@gmail.com", "Carla", "123456", "123456");

**3- Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

## 1.2 [RegInval]

Registro de Usuario con datos inválidos (repetición de contraseña invalida).

**1-Vamos al formulario de registro**

PO\_HomeView.*clickOption*(*driver*, "signup", "id", "btnRegistro");

**2- Rellenamos el formulario.**

PO\_RegisterView.*fillForm*(*driver*, "nerea@gmail.com", "Nerea", "123456", "111111");

PO\_View.*getP*();

**3- Comprobamos el error de contraseña no coincide.**

PO\_View.*checkElement*(*driver*, "text", "Las contraseñas no coinciden");

## 2.1 [InVal]

Inicio de sesión con datos válidos.

**1-Pinchamos en el botón de Indentidicate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2-Rellenamos el formulario.**

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3- Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

## 2.2 [InInVal]

Inicio de sesión con datos inválidos (usuario no existente en la aplicación).

**1-Pinchamos en el botón de Identifícate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2- Rellenamos el formulario.**

PO\_LoginView.*fillForm*(*driver*, "daniel@gmail.com", "123456");

**3- Comprobamos que seguimos en la vista de login**

PO\_View.*checkElement*(*driver*, "id", "vistaLogin");

PO\_View.*checkElement*(*driver*, "text", "Datos incorrectos");

3.1 [LisUsrVal]

Acceso al listado de usuarios desde un usuario en sesión.

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2-Rellenamos el formulario**.

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3- Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

**4-Pinchamos en la opción de menu de ver usuarios:**

List<WebElement> elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"listaUsuarios\"]");

elementos.get(0).click();

**5-Contamos el número de filas con usuarios**

List<WebElement> elemento = SeleniumUtils.*EsperaCargaPagina*(*driver*, "free", "//tbody/tr", PO\_View.*getTimeout*());

*assertTrue*(elemento.size() == 5);

**6-Ahora nos desconectamos**

PO\_RegisterPublicationView.*clickOption*(*driver*, "logout", "id", "btnIdentificate");

## 3.2 [LisUsrInVal]

Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión. Debe producirse un acceso no permitido a vistas privadas.

**1- Vamos directamente a la siguiente dirección**

*driver*.navigate().to("http://localhost:8090/user/list");

2-**Comprobamos que se nos redirige al login**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

## 4.1 [BusUsrVal]

Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión.

**1-Pinchamos en el botón d eIdentifícate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2- Rellenamos el formulario**.

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3-Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

**4-Pinchamos en la opción de menu de ver usuarios:**

List<WebElement> elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"listaUsuarios\"]");

elementos.get(0).click();

**5-Acceso al campo de busqueda**

elementos= PO\_View.*checkElement*(*driver*,"free","//\*[@id=\"campoBusqueda\"]");

elementos.get(0).click();

**6-Insertamos la cadena "mar"**

elementos.get(0).sendKeys("mar");

**7-Pinchamos en buscar**

elementos= PO\_View.*checkElement*(*driver*,"free","//\*[@id=\"btnBuscar\"]");

elementos.get(0).click();

**8-Contamos el número de filas con usuarios**

List<WebElement> elemento = SeleniumUtils.*EsperaCargaPagina*(*driver*, "free", "//tbody/tr", PO\_View.*getTimeout*());

*assertTrue*(elemento.size() == 2);

**9-Nos desconectamos**

PO\_RegisterPublicationView.*clickOption*(*driver*, "logout", "id", "btnIdentificate");

## 4.2 [BusUsrInVal]

Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado. Debe producirse un acceso no permitido a vistas privadas.

**1-Vamos directamente a la siguiente dirección**

*driver*.navigate().to("http://localhost:8090/user/list?searchText=mar");

**2-Comprobamos que se nos redirige al login**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

## 5.1 [InvVal]

Enviar una invitación de amistad a un usuario de forma valida

**1-Nos registramos coomo usuario “Pedro”, email: pedro@gmail.com**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**2**-**Vamos a la vista que lista todos los usuarios del sistema**

PO\_HomeView.*clickOption*(*driver*, "/user/list", "text", "Usuarios");

**3-Comprobamos que en los 5 primeros usuarios son Lucas, María, Marta, Pelayo y Andrea**

SeleniumUtils.*textoPresentePagina*(*driver*, "Lucas");

SeleniumUtils.*textoPresentePagina*(*driver*, "María");

SeleniumUtils.*textoPresentePagina*(*driver*, "Marta");

SeleniumUtils.*textoPresentePagina*(*driver*, "Pelayo");

SeleniumUtils.*textoPresentePagina*(*driver*, "Andrea");

**4-** **Click sobre botón de Pelayo con id=** resendButton13 **con mensaje “Agregar amigo”, al hacer click crea una petición hecha por Pedro con destinatario Pelayo. El texto del botón pasa a ser “Petición enviada”**

*assertEquals*("Agregar amigo",PO\_NavView.*getButtonText*(*driver*, "resendButton13"));

PO\_NavView.*clickButton*(*driver*, "resendButton13");

## 5.2 [InvInVal]

Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente. No debería dejarnos enviar la invitación, se podría ocultar el botón de enviar invitación o notificar que ya había sido enviada previamente.

**1-Nos registramos coomo usuario “Pedro”, email: pedro@gmail.com**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**2-Vamos a la vista que lista todos los usuarios del sistema**

PO\_HomeView.*clickOption*(*driver*, "/user/list", "text", "Usuarios");

**3**-**Comprobamos que en los 5 primeros usuarios son Lucas, María, Marta, Pelayo y Andrea**

SeleniumUtils.*textoPresentePagina*(*driver*, "Lucas");

SeleniumUtils.*textoPresentePagina*(*driver*, "María");

SeleniumUtils.*textoPresentePagina*(*driver*, "Marta");

SeleniumUtils.*textoPresentePagina*(*driver*, "Pelayo");

SeleniumUtils.*textoPresentePagina*(*driver*, "Andrea");

**4-Click sobre botón de Lucas con id=noresendButton4, como ya le hemos enviado una petición y no ha contestado, el botón tiene el texto “Petición enviada” y no tiene ninguna funcionalidad**

*assertEquals*("Petición enviada",PO\_NavView.*getButtonText*(*driver*, "noresendButton4"));

PO\_NavView.*clickButton*(*driver*, "noresendButton4")

**5-El botón sigue igual**

*assertEquals*("Petición enviada",PO\_NavView.*getButtonText*(*driver*, "noresendButton4"));

## 6.1 [LisInvVal]

Listar las invitaciones recibidas por un usuario, realizar la comprobación con una lista que al menos tenga una invitación recibida.

**1-Nos registramos coomo usuario “Pedro”, email: pedro@gmail.com**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**2**-**Vamos a la vista que lista las invitaciones recibidas**

PO\_HomeView.*clickOption*(*driver*, "/request/list", "text","Listado de invitaciones recibidas:");

**3-Comprobamos que los usuarios que han mandado invitaciones a Pedro son Andrea, María, José, Sansa y Cersei**

SeleniumUtils.*textoPresentePagina*(*driver*, "María");

SeleniumUtils.*textoPresentePagina*(*driver*, "José");

SeleniumUtils.*textoPresentePagina*(*driver*, "Sansa");

SeleniumUtils.*textoPresentePagina*(*driver*, "Cersei");

SeleniumUtils.*textoPresentePagina*(*driver*, "Jon");

## 7.1 [AcepInvVal]

Aceptar una invitación recibida.

**1-Nos registramos como usuario “Pedro”, email: pedro@gmail.com**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**2**-**Vamos a la vista que lista las invitaciones recibidas**

PO\_NavView.*clickOption*(*driver*, "/request/list", "text", "Listado de invitaciones recibidas:");

**3-Comprobamos que los usuarios que han mandado invitaciones a Pedro son Andrea, María, José, Sansa y Cersei**

SeleniumUtils.*textoPresentePagina*(*driver*, "Andrea");

SeleniumUtils.*textoPresentePagina*(*driver*, "María");

SeleniumUtils.*textoPresentePagina*(*driver*, "José");

SeleniumUtils.*textoPresentePagina*(*driver*, "Sansa");

SeleniumUtils.*textoPresentePagina*(*driver*, "Cersei");

**4-Aceptamos invitación de Andrea**

PO\_NavView.*clickButton*(*driver*, "acceptFriendButton24");

SeleniumUtils.*esperarSegundos*(*driver*, 2);

**5-Comprobamos que la invitación de Andrea ya no está**

SeleniumUtils.*textoNoPresentePagina*(*driver*, "Andrea");

## 8.1 [ListAmiVal]

Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo.

**1-Vamos al formulario de registro**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2-Rellenamos el formulario.**

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3-Pinchamos en la opción de menu de ver usuarios:**

PO\_NavView.*clickOption*(*driver*, "/friend/list", "text", " Listado mis amigos");

SeleniumUtils.*textoNoPresentePagina*(*driver*, "Cersei");

PO\_NavView.*clickOption*(*driver*, "/request/list", "text", "Listado de invitaciones recibidas:");

SeleniumUtils.*textoPresentePagina*(*driver*, "Cersei");

**4-Aceptar peticion de Cersei**

PO\_NavView.*clickButton*(*driver*, "acceptFriendButton28"); //botón de Cersei

SeleniumUtils.*esperarSegundos*(*driver*, 2);

SeleniumUtils.*textoNoPresentePagina*(*driver*, "Cersei");

PO\_NavView.*clickOption*(*driver*, "/friend/list", "text", " Listado mis amigos ");

**5-Vuelta para comprobar que cersei es amiga**

SeleniumUtils.*textoPresentePagina*(*driver*, "Cersei");

## 9.1 [PubVal]

Crear una publicación con datos válidos.

**1-Pinchamos en el botón Identifícate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2- Rellenamos el formulario.**

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3-Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

**4-Pinchamos en la opción de menu de ver usuarios:**

List<WebElement> elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"publicaciones-menu\"]/a");

elementos.get(0).click();

**5- Pinchamos en crear publicación**

elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"crearPublication\"]");

elementos.get(0).click();

SeleniumUtils.*esperarSegundos*(*driver*, 5);

**6-Rellenamos los campos**

PO\_RegisterPublicationView.*fillFormAddPublication*(*driver*, "prueba1", "Esto es una prueba");

**7-Comprobamos que estamos en la vista con lista de publicaciones**

PO\_View.*checkElement*(*driver*, "id", "listaPublicaciones");

**8-Comprobamos que aparece la publicación añadida**

elementos = PO\_View.*checkElement*(*driver*, "text", "prueba1")

## 10.1 [LisPubVal]

Acceso al listado de publicaciones desde un usuario en sesión.

**1-Pinchamos en el botón Identifícate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2-Rellenamos el formulario**.

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3- Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

**4-Pinchamos en la opción de menu de publicaciones:**

List<WebElement> elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"publicaciones-menu\"]/a");

elementos.get(0).click();

**5-Pinchamos en mis ublicaciones**

elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"misPublicaciones\"]");

elementos.get(0).click();

**6-Comprobamos que aparece la publicacion Publicacion1**

elementos = PO\_View.*checkElement*(*driver*, "text", "Publicacion1");

## 11.1 [LisPubAmiVal]

Listar las publicaciones de un usuario amigo

**1-Pinchamos en el botón Identifícate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2-Rellenamos el formulario**.

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3- Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

**4-Pinchamos en la opción de menu de publicaciones recibidas:**

List<WebElement> elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"peticionesRecibidas\"]");

elementos.get(0).click();

**5-Pinchamos en el boton de aceptar amistad**

SeleniumUtils.*esperarSegundos*(*driver*, 1);

elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"acceptFriendButton25\"]");

elementos.get(0).click();

**6-Pinchamos en la opcion del menu mis amigos**

elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"misAmigos\"]");

SeleniumUtils.*esperarSegundos*(*driver*, 1);

elementos.get(0).click();

SeleniumUtils.*esperarSegundos*(*driver*, 1);

**7-Pinchamos en el nombre "andrea" para ver sus publicaciones**

elementos = PO\_View.*checkElement*(*driver*, "free", "//\*[@id=\"nombreAmigo\"]");

elementos.get(0).click();

**8-Comprobamos que aparece la publicacion Publicacion5**

elementos = PO\_View.*checkElement*(*driver*, "text", "Publicacion5");

## 11.2 [LisPubAmiInVal]

Utilizando un acceso vía URL tratar de listar las publicaciones de un usuario que no sea amigo del usuario identificado en sesión.

**1-Pinchamos en el botón Identifícate**

PO\_HomeView.*clickOption*(*driver*, "login", "id", "btnIdentificate");

**2-Rellenamos el formulario.**

PO\_LoginView.*fillForm*(*driver*, "pedro@gmail.com", "123456");

**3-Comprobamos que entramos en la sección privada**

PO\_View.*checkElement*(*driver*, "id", "zonaPrivada");

**4-Intentamos acceder a las publicaciones de un usuario que aun no es nuestro amigo**

*driver*.navigate().to("http://localhost:8090/publication/publicationsFriend/18");

## 12.1 [PubFot1Val]

Crear una publicación con datos válidos y una foto adjunta.

## 12.1 [PubFot2Val]

Crear una publicación con datos válidos y sin una foto adjunta.

## 13.1 [AdInVal]

Inicio de sesión como administrador con datos válidos.

## 13.2 [AdInInVal]

Inicio de sesión como administrador con datos inválidos (usar los datos de un usuario que no tenga perfil administrador).

## 14.1 [AdLisUsrVal]

Desde un usuario identificado en sesión como administrador listar a todos los usuarios de la aplicación.

## 15.1 [AdBorUsrVal]

Desde un usuario identificado en sesión como administrador eliminar un usuario existente en la aplicación.

## 15.2 [AdBorUsrInVal]

Intento de acceso vía URL al borrado de un usuario existente en la aplicación. Debe utilizarse un usuario identificado en sesión pero que no tenga perfil de administrador.

# Cualquier otra información necesaria para una descripción razonablemente detallada de lo entregado y su correcto despliegue y ejecución.

**Caso de uso 11**

Para comprobar el correcto funcionamiento de este caso de uso, dado que tal y como está la base de datos algunos usuarios no tienen amigos, antes de ir a la opción “Mis amigos” y comprobar susu publicaciones se deberían seguir los siguientes pasos:

1. Nos identificamos por ejemplo como [pedro@gmail.com](mailto:pedro@gmail.com)
2. Vamos a “Peticiones recibidas” y aceptamos la petición de María
3. Vamos a “Mis amigos”
4. Pinchamos sobre el nombre María y así podemos ver sus publicaciones.

La prueba de test “ListAmiVal” obligó a hacer una pequeña modificación debido a un error de aserción. En los métodos está definido que no puede haber el mismo texto 2 veces, por lo que en la vista “friend/list.html” se ha modificado el título de “Mis amigos” a “Listado de mis amigos” (coincidía con el botón de navegación “Mis amigos”)